Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

**Algoritmos Avançados**

2020/2021 — 1º Semestre

# 1º Trabalho — Estratégias de Desenvolvimento de Algoritmos

Data limite de entrega: 7 de dezembro de 2020

## Hipótese A – Pesquisa Exaustiva

Após escolher um dos problemas seguintes, desenvolva e teste um algoritmo de pesquisa exaustiva para o resolver.

De seguida, analise o esforço computacional realizado pelo algoritmo desenvolvido. Para isso deve:

a) Analisar a complexidade do algoritmo.

b) Realizar uma sequência de testes com instâncias sucessivamente maiores do problema. E registar e analisar: (1) o número de operações básicas efectuadas, (2) o tempo de execução e (3) o número soluções / configurações testadas.

c) Comparar os resultados obtidos nos dois passos anteriores.

d) Estimar o tempo de execução requerido para instâncias do problema de muito maior dimensão.

e) Escrever um relatório sucinto (máx. 6 págs.).

1 – Determinar a (uma) **clique de cardinalidade máxima** de um dado grafo não orientado $G$, com $n$ vértices e $m$ arestas. Um subconjunto dos vértices de $G$ diz-se uma **clique de *G*** se entre quaisquer dois vértices distintos da clique existir uma aresta em $G$. Ou seja, a clique induz um subgrafo completo em $G$.

2 – Determinar o (um) **conjunto independente de vértices de cardinalidade máxima** de um dado grafo não orientado $G$, com $n$ vértices e $m$ arestas. Um subconjunto dos vértices de $G$ diz-se um **conjunto independente de vértices de *G*** se entre quaisquer dois vértices distintos do conjunto não existir uma aresta em $G$.

3 – Determinar a (uma) **cobertura de vértices de cardinalidade mínima** de um dado grafo não orientado $G$, com $n$ vértices e $m$ arestas. Um subconjunto dos vértices de $G$ diz-se uma **cobertura de vértices de *G*** se cada aresta de $G$ for incidente em (pelo menos) um vértice pertencente à cobertura de vértices.

4 – Determinar o (um) **conjunto dominante de cardinalidade mínima** de um dado grafo não orientado $G$, com $n$ vértices e $m$ arestas. Um subconjunto dos vértices de $G$ diz-se um **conjunto dominante de *G*** se cada vértice não pertencente ao conjunto dominante for adjacente a (pelo menos) um vértice pertencente ao conjunto dominante.

5 – Determinar a (uma) **cobertura de arestas de cardinalidade mínima** de um dado grafo não orientado *G*, com *n* vértices e *m* arestas. Um subconjunto das arestas de *G* diz-se uma **cobertura de arestas de *G*** se cada vértice do grafo for incidente em (pelo menos) uma aresta pertencente à cobertura de arestas.

6 – Determinar o (um) **conjunto dominante de arestas de cardinalidade mínima** de um dado grafo não orientado *G*, com *n* vértices e *m* arestas. Um subconjunto das arestas de *G* diz-se um **conjunto dominante de arestas de *G*** se cada aresta não pertencente ao conjunto dominante for adjacente a (pelo menos) uma aresta pertencente ao conjunto dominante.

7 – Determinar o **número cromático** de um dado grafo não orientado *G*, com *n* vértices e *m* arestas. O **número cromático de *G*** expressa o número **mínimo** de cores necessárias para colorir os **vértices** de *G*, de tal modo que a vértices adjacentes correspondam cores distintas.

8 – Determinar o **índice cromático** de um dado grafo não orientado *G*, com *n* vértices e *m* arestas. O **índice cromático de *G*** expressa o número **mínimo** de cores necessárias para colorir as **arestas** de *G*, de tal modo que a arestas adjacentes correspondam cores distintas.

9 – **The Bottleneck Traveling Salesperson Problem** (Bottleneck TSP) – Considere a seguinte modificação do Problema do Caixeiro-Viajante: dado um grafo não orientado *G(V, E)* determinar, se existir, um (o) ciclo Hamiltoniano que minimiza a distância associada à aresta mais longa do ciclo.

**Hipótese B – Dynamic Programming**

Pretende-se avaliar o desempenho computacional de algoritmos de Programação Dinâmica para diferentes problemas.

Assim, após escolher um dos temas de trabalho seguintes, e com base na mesma estratégia de resolução para esse problema, desenvolva e teste:

- um algoritmo **recursivo**,
- um algoritmo recursivo usando ***"memoization"***,
- um algoritmo de **Programação Dinâmica**.

Deverá ser feita uma análise da eficiência computacional e das limitações dos algoritmos desenvolvidos.

Para isso deve:

a) Analisar a sua complexidade.

b) Realizar um conjunto de testes com instâncias de dados sucessivamente maiores.

c) Registar e analisar (1) o tempo de execução e (2) o número operações básicas realizadas.

d) Estimar o tempo de execução requerido para instâncias de muito maior dimensão.

e) Comparar o desempenho dos três algoritmos.

f) Escrever um relatório sucinto (máx. 6 págs.).

10 – Dadas duas cadeias de caracteres, determinar a sua **subcadeia comum mais longa** ("The Longest Common Substring Problem").

11 – Dadas duas cadeias de caracteres, determinar a sua **subsequência comum mais longa** ("The Longest Common Subsequence Problem").

12 – Dadas duas cadeias de caracteres, determinar a sua **distância** usando o **Algoritmo de Wagner-Fischer** ("The String to String Correction Problem").

13 – Dadas duas sequências de caracteres, efetuar o seu **alinhamento** usando o **Algoritmo de Needleman-Wunsch** ("The Sequence Alignment Problem").

14 – Dadas duas sequências de caracteres, efetuar o seu **alinhamento** usando o **Algoritmo de Smith-Waterman** ("The Sequence Alignment Problem").

15 – Determinar o **fecho transitivo** de um digrafo, usando o **Algoritmo de Warshall**.

16 – Determinar o **caminho mais curto entre cada par de vértices** de um grafo, usando o **Algoritmo de Floyd** ("The All-Pairs Shortest-Path Problem").

17 – Dado um grafo com $n$ vértices e $m$ arestas, em que os vértices representam as cidades e as arestas representam as respetivas distâncias, determinar o (um) percurso ótimo para um **caixeiro-viajante** que pretenda visitar essas $n$ cidades ("The Traveling Salesperson Problem").

18 – Determinar o **menor número de moedas** para um dado troco, i.e., que perfazem um dado valor ("The Change-Making Problem").

19 – Resolver o **Problema da Partição**: dado um conjunto $S$ de números inteiros e positivos, é possível subdividir $S$ em dois subconjuntos disjuntos, $S_1$ e $S_2$, de tal modo que a soma dos elementos de $S_1$ é igual à soma dos elementos de $S_2$? **Nota:** trata-se de um problema de decisão NP-Completo, mas que é resolúvel usando um algoritmo (pseudo-polinomial) de programação dinâmica.

J. Madeira, 10 de novembro de 2020